RGeostats Manual

D. Renard, F. Ors

June 20th, 2014

Abstract

This document constitutes the users manual for the package RGeostats. It gives an overall presentation of the package, developed using R language. For a more detailed description of each function, the reader will refer to its on-line documentation. Finally some tutorials are also available in the standard RGeostats distribution which enable the interested user to run some examples on provided data sets. You should refer to the Getting Started manual for installation of RGeostats package.

Part I

History of the RGeostats package

The Centre de Géostatistique of the Ecole des Mines de Paris spent several years developing different commercial libraries or softwares in the past. Let us mention:

- *GEOSLIB*: the first geostatistical library in FORTRAN
- *BLUEPACK*: a geostatistical package that lasted over 10 years and was famous in most mining and oil companies over the world
- *SIMPACK*: a package dedicated to geostatistical stochastic simulations
- *HERESIM*: a package, developed jointly with Institut Français du Pétrole, based on the Plurigaussian simulation technique
- *ISATIS*: the geostatistical toolbox, developed jointly and commercialized by Géovariances

It is therefore a tradition for the Centre de Géostatistique to imagine, produce and commercialize the algorithms developed by scientists so that practitioner

can apply these fancy techniques to the different fields in their own domain of activity, without having to bother writing lines of code.

However, these packages do not allow the user to modify the code in order to test new ideas or algorithms. This is the reason why, starting in late 1990's, some researchers started introducing some algorithms in the framework of the R package. This was initiated with the *GEFA* package developed for the fisheries community, which uses geostatistical techniques to forecast the fish density by species and age. The user could benefit from all the advantages due to the large number of contributors of R developers, combined to the procedures established by the researchers of the Centre de Géostatistique.

As usual with writing packages using the R interpreted language, the GEFA package needed to be strongly improved for improving the calculation speed. This usually involves writing pieces of the package in using a compiled language (such as C or C++).

For that reason, the package *RGeoS* was created in the year 2001 containing a set of R objects to manipulate data, parameters and results. The package *RGeoS* is based on a library of geostatistical code written in C (and C++) called *Geoslib*.

Recently, the package *RGeoS* has been renamed ***RGeostats*** during the year 2014 to better explain its contents and avoid conflict with packages with similar names.

The main characteristics of the RGeostats package is to perform geostatistical operations simultaneously on several (p) variables in a space of any dimension (n): we will designate this package as R_n-R_p package. However, some techniques are not defined for any space dimension, nor any number of variables treated simultaneously: a special test restricts their usage.

1 What is RGeostats ?

RGeostats is provided as a binary R package for Windows, Linux and Mac platform . It provides:

- all the R procedures with the corresponding on-line help (use the command *?func_name* to access the on-line help of the function *func_name*)
- the object library of the geostatistical code Geoslib
- some demonstration case studies: the user can run them using the command demo() with the available data sets.

Note that the *source code corresponding to the Geoslib library is not available.*

2 Who can use RGeostats ?

RGeostats can be downloaded by anyone.

RGeostats can be used free of charge in a non-commercial use.

3 The reference to RGeostats

When you use this software for publication, please use the following reference:

Renard D., Bez N., Desassis N., Beucher H., Ors F., Laporte F.
RGeostats: The Geostatistical package [version number].
MINES ParisTech.
Free download from: http://cg.ensmp.fr/rgeostats

4 Where can I find RGeostats ?

The package RGeostats must be downloaded from the web site of the Centre de Géostatistique of the Ecole des Mines de Paris:

http://cg.ensmp.fr/rgeostats

This site contains several directories, such as:

- the user community Board where you can:
  - Download RGeostats package (according to the Operating System where you want to use the package): this operation requires that you register to the Board first
  - Ask any question about any issue you may encounter
  - Learn on how to use specific parts of the package by reviewing the corresponding Tutorial
- the Documentation directory where you can find several case studies: each case study contains:
  - a PDF file where the case study is fully described
  - the ASCII file(s) that are used in the case study: however, these data sets are already contained in the distribution and can be loaded using the data() procedure.
- the Demo directory where you can find several demonstration scripts

- the Function directory where you can find the on-line help for all functions

The package is provided for few platforms. For each platform, RGeostats is provided as a single file in an archive format. The extension of the archive file depends upon the platform:

- *Windows* 32 or 64 bits: file with extension *zip*
- *LINUX 32-bits*: file with extension *linux32.tar.gz*
- *LINUX 64-bits*: file with extension *linux64.tar.gz*
- *Mac*: file with extension *tgz*

5 Installing the RGeostats package

5.1 Installing R

The package R must be installed first. R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. This package can be downloaded from the site:

http://cran.r-project.org

If available for your Operating System, it is easier to install directly the dedicated binary version. Otherwise, one can always download the source code, configure it and compile it. Then please follow official information provided on the site.

The installation requires the Administrator rights.

5.2 Required package

The package Rcpp is required and can be downloaded from the CRAN web site.

5.3 Additional contributions

Moreover, some additional contributions can be downloaded (from the same site): such as *maps* and *mapproj* which are only necessary in some parts of the package *RGeostats* and will be only needed upon request.

Each extension comes as an archive file.

5.4 Installing an additional contribution

When installaing a package, one may choose between:

- installing it as a permanent extension of R: this operation requires the Administrator rights as the RGeostats add-on package is written on the directory where R distribution is installed. The installation is performed by typing:

  R CMD INSTALL mypack

- installing it as a personal extension: this is the case when an extension often varies. This installation does not require the Administrator rights. The package is installed on a user's dedicated directory (say *my_dir*) by typing:

  R CMD INSTALL mypack --library=my_dir

6 Getting started with RGeostats

6.1 Loading the package

You must first start R in a working directory. You may have one working directory by project. You launch R by typing the corresponding command (or clicking the corresponding icon on Windows for example)

Within the R session, you must load the RGeostats. If RGeostats has been installed in the R distribution directory, simply type:

```
library(RGeostats)
```

Otherwise type:

```
library(RGeostats,lib.loc=/my_dir)
```

This information can be stored in a specific (hidden) file, called *.First*, which is automatically started each time R is loaded in the working directory.

In order to create it, the best solution is to enter the R session and to define it interactively by typing:

```
fix(.First)
```

The previous command launches a text editor. The name of the text editor can also be parametrized in the *.First* file for future use.

The contents of the .First file could be something as :

```
.First
function ()
{
library(RGeostats, lib.loc="/my_dir")
}
```

6.2 Additional information on RGeostats

When RGeostats is loaded successfully, the user can check the version of the RGeostats package. This information may become usefull for further discussion concerning the ability of the package to perform a given task or to describe a mysfunctioning:

```
acknowledge("RGeostats")
```

The following message is displayed (which may evolve with time):

```
Package RGeostats (Version:XX.X.X - Date:mm,dd,yy)
==================================================
Geoslib Library (Version:XX.X.X - Date:mm,dd,yy)
Authors:
Didier RENARD      (didier.renard@mines-paristech.fr)
Nicolas BEZ        (nicolas.bez@ird.fr)
Nicolas DESASSIS   (nicolas.desassis@mines-paristech.fr)
Helene BEUCHER     (helene.beucher@mines-paristech.fr)
Fabien ORS         (fabien.ors@mines-paristech.fr)
Florence LAPORTE
```

Another interesting function (R standard) gives the position where the package has been loaded by typing:

```
search()
```

The following information is obtained in the R session (the contents depends upon the R version, the user's environment and the list of packages already loaded):

```
 [1] ".GlobalEnv"        "package:RGeostats" "package:Rcpp"
 [4] "package:stats"     "package:graphics"  "package:grDevices"
 [7] "package:utils"     "package:datasets"  "package:methods"
[10] "Autoloads"         "package:base"
```

The order of the loaded packages may vary depending on the user's preferences. It is easy to see that here RGeostats is loaded in position 2.

The user can then type the following command in order to get the list of all the procedures included in RGeostats:

```
ls(pos=2)
```

Another way to learn about each command (say my_command) is to ask for its calling arguments by typing:

```
args(my_command)
```

But obviously the best solution is to get the information on the command by typing:

```
?my_command
```

The information can even be displayed in a more sophisticated manner is the user has launched a HTML browser beforehand by typing the following command at least once in the R session:

```
help.start()
```

Part II

Description of the Package RGeostats

7 Organization

The RGeostats package offers a set of objects. They belong to classes according to the S4 mechanism (defined in the methods package). Each geostatistical procedure in RGeostats uses these objects as input arguments (together with some additional parameters); some procedures produce one object as output.

Each class has some generic methods, some accessors and some specific utilities attached. A generic method is a function which has an implementation for an object of any class. An accessor is a specific function which gives access (read or write) to the different items of the object of any class. A utility is a function designed specifically to address an object of a given class.

8 Classes

The different classes of RGeostats are characterized by their names. To get more information on objects of one class, the user can type the command*:*

```
class?class_name
```

where *class_name* is the name of one of the previous classes: for example *class?db*. The list of classes follows:

db: data base containing the input data or the output results

vario: experimental spatial characteristics calculated from data, such as experimental variograms, covariances, generalized variograms

model: model describing the spatial characteristics, such as the variogram, the covariance or the generalized covariance model

neigh: set of parameters describing the selection of samples used for processing a target point, called neighborhood.

anam: set of parameters used to transform a sampled variable from its initial distribution to a gaussian standardized law, and vice-versa.

rule: the lithotype rule used to convert one or two gaussian random functions into a categorical variable (facies) through thresholds

thresh: a set of intervals used to convert a variable into a categorical variable or a set of indicators

polygon: a set of one or several polysets. Each polyset is a closed broken line defined in 2-D.

tokens: a set of object definitions used for object base simulations

These major classes also use the following auxiliary sub-classes (that the user should ignore):

vardir: a directional experimental variogram. Any object of the *vario* class is a set of one or several objects of the vardir class.

melem: a basic covariance structure. Any object of the *model* class is a set of one or several objects of the *melem* class.

polyset: a 2-D broken line. Any object of the *polygon* class is a set of one or several objects of the *polyset* class.

9 Accessors

An accessor is an operator which enables the user to read or write one field of an object belonging to a given class, using a simple syntax, without having to know anything concerning the internal structure of the class. There are four generic accessors, characterized by their symbol:

$: to read the value of a slot of an object (value read accessor)

$<-: to assign a value to a slot of an object (value write accessor)

[: to read the value of a slot of an object which is considered as an array (array read accessor)

[<-: to assign a value to a slot of an object which is considered as an array (array write accessor)

The accessors will be illustrated here using the example of the *db* class (which will be described more exhaustively in subsequent paragraph).

An accessor is designed in order to question the dimension of the space in which the *db* is established:

```
db$ndim
```

Another accessor gives access to the coordinates of the grid origin:

```
db$x0
```

Note that, as such an accessor is only valid in the case of a grid, it will return an error is used for a *db* not organized as a grid.

An object of the *db* class stores the data values within the field *items,* which consists in a data frame: the columns corresponds to the fields and the rows to the samples. Therefore if we want to access to the value of the third variable for the second sample, we should use the syntax:

```
db@items[2,3]
```

Instead, using the read accessor, we will use the equivalent accessor:

```
db[2,3]
```

10 Generic methods

Any object belonging to a class has a set of generic methods attached according to the S4 mechanism. To get more information on these generic methods, use the command:

```
methods?method_name
```

where *method_name* corresponds to the name of the generic function. For example:

```
methods?plot
```

Some new generic functions and pseudo-generic functions have been added in the RGeostats package.

Several generic methods are defined (the same method name can be used for other objects belonging to other packages). These generic functions use their first argument as a *signature* in order to decide upon the exact function that will be triggered: this signature consists of an object of a given *RGeostats* class.

show: display the contents of an object belonging to a class

print: display the contents of an object belonging to a class. It may give similar results as the *show* generic function, but is used sometimes differently for specific printouts.

plot: displays the contents of an object belonging to a class

Some generic methods have been added, although they are specific to RGeostats objects only:

ascii.write: dumps the contents of an object belonging to a class in an ASCII file according to a specific format. The format is explained separately together with the class description.

In the following example (where *dbobj* stands for an object of the class *db*) the following command:

```
plot(dbobj,...)
```

automatically launches the plot command for the db class (command db.plot). Therefore, the previous command is exactly similar to:

```
db.plot(dbobj,...)
```

This mechanism is essential in order to understand that the same generic function can have different arguments according to the object to which it is applied.

Finally, some additional functions have been added that can be considered as *pseudo-generic*: the first argument cannot be an RGeostats (already existing) object as we are precisely creating it. Instead, the user must provide the name of the class as a signature. Otherwise the signature is asked interactively.

The following command gives an example of a pseudo-generic function used to read an object of the class *db* from the ASCII file called *myfile*. The resulting object will be stored in the RGeostats object called *mydb*:

```
mydb <- ascii.read(signature=db,filename=myfile)
```

Note that the command *methods?method_name* is not valid for the pseudo-generic commands. In the following, we will make no difference between generic and pseudo-generic methods.

The following pseudo-generic methods have been added:

ascii.read: read the contents of an object belonging to a class from an ASCII file according to a specific format.

digitize: digitize an object from a graphic plot

11 ASCII format

Each class has a method for reading or writing the contents of an object which belongs to this class in an ASCII file. The format is obviously specific to each class.

Several features are common to these methods, whatever the class:

- the values (numeric or alphanumeric) are separated by blank spaces. They may be coded on any number of lines: the line change is not significant.
- an alphanumeric variable may not contain any blank, unless the variable is enclosed within quotes.
- comments can be inserted anywhere: a comment starts with the # character and extends until the end of the line.
- a missing numeric value is replaced by the string -999. Note that the exact spelling must be used (including the final decimal point).

12 Classes

This paragraph gives more information for each class of the RGeostats package. It describes systematically:

- the different fields contained in an object of the class
- the syntax of the different accessors
- the generic functions
- the utilities
- the other functions, specific to each class, are merely described in this manual. The user should refer to their on-line manual for more information.

12.1 Db

This class is used to store input data set or output results. It corresponds to a set of columns (also called *fields*) defined on a set of *samples*. The variables are numeric only and stored as real values (even if they can be printed in integer format). The samples can be either organized as a regular grid or non organized (set of isolated points).

12.1.1 Fields

The Db class contains the following slots:

flag.grid: tells if the data base is organized as a grid or not

ndim: Dimension of the space

If the data base is organized as a regular grid:

x0: array (dimension: ndim) which gives the coordinates of the grid origin (lowest values for each coordinate)

dx: array (dimension: ndim) which gives the extension of the grid mesh in all space direction

nx: array (dimension: ndim) which gives the number of grid nodes along each axis

Note that if cells are attached to grid nodes (in the case of the estimation of the average target variable over blocks for example), each cell is conventionally centered on the grid node.

The array of locators (dimension: number of fields). A locator is a specific identifier followed by its rank (starting from 1). A locator indicates the role that a given field for the data base plays. The list of identifiers follows:

- x: coordinate
- z: data variable (on which the actual calculations are processed)
- v: measurement error variance
- f: auxiliary variable used as external drift
- g: gradient components
- l : lower bounds for intervals
- u: upper bounds for intervals

- p: proportion for categorical variables (facies)

For example, the field attached to the locator 'x1' gives the first coordinates of the samples contained in the data base; locator 'f2' gives the second external drift. There is no limitation for the rank as RGeostats is designed for any number of variables and any space dimension.

There are some other locators which can only be either present or absent (there is no rank):

- w: weighting variable
- code: code variable
- sel: the selection

12.1.2 Accessors

These are the different read/write accessors of some attributes in the class:

db$flag.grid: the grid status

db$ndim: the space dimension

db$x0: the coordinates of the grid origin. If the data base is not organized as a grid, NA is returned when reading, an error is issued when writing.

db$dx: the meshes of the grid. If the data base is not organized as a grid, NA is returned when reading, an error is issued when writing.

db$nx: the number of grid meshes. If the data base is not organized as a grid, NA is returned when reading, an error is issued when writing.

db$locators: the list of locators for the different fields of the data base.

db$names: the list of names for the different fields of the data base.

db$items: the set of values for the different fields of the data base (produced as a data.frame)

db$nech: number of samples. It is not defined when writing.

db$natt: number of variables. It is not defined when writing.

These are the different read/write accessors of some arrays in the class:

db[i,j]: the value for the field j for the sample i

db[i,]: the values for all fields for the sample i

db[,j]: the values of field j for all samples

db[,]: all the values of the data base (gives the same result as the command *db[]* or *db$items*)

Errors are issued if the rank of the field or the rank of the sample is erroneous.

12.1.3 Generic functions

db.digit: Digitize a point location from a graphic screen. If a data base is passed as argument, return the characteristics of the sample (from a Regular Grid or the Set of Points) closest to the digitized point. This function corresponds to the generic command *digitize*.

db.plot: Represent the contents of a data base graphically. This function corresponds to the generic command *plot*.

db.print: Print the contents of a data base. This function corresponds to the generic command *print*.

db.read: Create a data base by reading an ASCII file according to the format specific to RGeostats. This function corresponds to the generic command *ascii.read*.

db.write: Write the contents of a data base into an ASCII file according to the format specific to RGeostats. This function corresponds to the generic command *ascii.write*.

12.1.4 Utilities

These utilities are specific to the Db class. They are merely described in this manual. The interested user will use the on-line help for more information:

db.add: Add new fields. This command also allows the creation of new variables as the on-the-fly transformation of already existing variables: this command is considered as the only *transformation method* within the RGeostats package. It can also be used to specify the locators of the newly created fields

db.compare: Calculate statistics (count, mean,variance, standard deviation, minimum, maximum) between different variables within the same data base.

db.create: Create a new data base, assigning its type (Regular Grid or Set of Points) and defining its general characteristics (space dimension, grid characteristics, number of points, ...).

db.delete: Delete an already existing data base.

db.edit: Edit the contents of the data base. This command does not include any feature for adding a new variable: you should use the command *db.add* beforehand.

db.extract: Retrieve one or several variables from a data base into a separate *data.frame* structure.

db.getcol: Return the rank a variable within the data base, from its locator type and rank.

db.getname: Return the name of a variable within the data base, from its locator type and rank.

db.grid.init: Create a new data base, organized as a Regular Grid, tailored from an already existing input data base. The new Grid covers the initial data base in all the directions of the space.

db.grid.locate: Return the absolute grid node located close to a point whose coordinates are given as input arguments.

db.indicator: Create new variables in the data base, corresponding to the indicators of the variable (corresponding to a z locator), given the set of threshold (passed as argument as an object belonging to the *thresh* class).

db.info: Return a list of data base characteristics

db.locate: Set the locator for one or several variables within a data base

db.normalize: Normalize a set of variables defined by their field numbers within the data base

db.polygon: Use an input polygon to select the samples of the data base (Regular Grid or Set of Points) which are included within at least one of the polysets constituting the Polygon (passed as argument as an object of the *polygon* class)

db.projec: Apply the projection of a data base. The characteristics of the projection must be defined beforehand using the command *projec.define*.

db.read.format: Create a data base organized as a Regular Grid by reading the contents of an ASCII file, according to the format BMP or ZYCOR.

db.rename: Change the name of a variable within a data base

db.rule: Convert a numerical variable into a new categorical variable according to a Lithotype Rule (passed as argument as an object of the *rule* class) and a set of proportions (either passed as an argument if stationary or contained in proportion variables of the data base otherwise).

db.selcombine: Combine the currently created selection to a selection already existing within the data base using a logical operation (or, and, not, xand, xor).

db.sel: Create a selection variable. This command also allows the creation of a new selection as the on-the-fly transformation of already existing variables.

db.start: Attach a data base to be used by a C code. This function should only be used prior to a call to C code which expects a data base: the data base will be connected using the slot number returned by the function *db.start*.

db.stat: Calculate the statistics (count, mean, variance, correlation, minimum, maximum, ...) or one or several fields of the data base.

db.thresh: Convert a numerical variable into a categorical variable using the thresholding information (passed as argument as an object of the *thresh* class).

db.write.format: Write the contents of a data base in an ASCII file, according to one of the following formats: CSV, IRAP or ZYCOR (the two last format are limited to Regular Grid data bases).

12.1.5 ASCII format

Format of the ASCII file for an object of the *db* class :

- A header line containing the list of locators (see the corresponding definition in the *Fields* paragraph)

If the object is a grid file, the header contains:

- the number of grid meshes in each direction
- the coordinate of the grid origin (lower left corner) in each direction
- the value of the grid mesh in each direction

The set of values:

- the set of real values corresponding to the different variables measured at the samples. We usually consider one sample per line. The set of real values is optional in the case of Grid Db.

12.2 Vario

The Vario class contains the information of the experimental variogram calculated from one or several variables contained in a data base. The term *variogram* is generic as it covers the following structural tools:

- Variogram
- Covariance (Centered)
- Transitive Covariogram
- Madogram (Variogram of order 1)
- Rodogram (Variogram of order 1/2)
- Variogram of a Poisson-weighted variable

If the data base contains several variables (with locator z), the corresponding experimental structural tools are then multivariate: we obtain several cross-variograms instead of a variogram for example.

Finally, the experimental structural tools can be calculated in several directions: samples are compared as long as they belong to a given direction, up to an angular tolerance.

12.2.1 Fields

The Vario class contains the following slots:

calcul: the type of structural tool

by.sample: the way the calculations are performed. When set to TRUE, a sample (seed) is selected and the corresponding sample-variogram is calculated by comparing the seed to any other sample according to distance and angular selection criteria. The final variogram is obtained as the average of the sample-variograms. When set to FALSE, the variogram is calculated in the classical way.

ndim: Space dimension.

nvar: Number of variables used for the calculation of the experimental structural tool.

opt.code: Option concerning the use of the code during the calculation of the experimental structural tool:

0: samples are compared whatever the value of the code (if defined),

1: two samples can be compared if the code variable is defined and the difference of the code values (at samples) is smaller than the parameter *tolcode*,

2: two samples can be compared if the code variable is defined and the code values (at samples) are different

scale: Scale value

tolcode: Tolerance for code, used when *opt.code* is defined and equal to 1

means: Array of the means of the variables under consideration. They are used for the Poisson variogram calculation

vars: Array of variances of the variables under consideration. They are used for the graphic representations

vardirs: An array of direction variograms (see below).

The experimental *directional* variogram Vardir contains the following slots:

npas: Number of calculation lags for the calculation in a direction.

npatot: Number of calculation values. This number is equal to npas for symmetric structural tools (variogram) and equal to 2*npas+1 for asymmetric structural tools (covariance).

pas: Lag value.

tol: Tolerance on the distances, defined as a percentage of the lag value.

flag.regular: Flag set to 1 if the lags are defined regularly as multiples of the parameter *pas*. When set to 0, the lags are calculated using the parameter *breaks* as thresholds on distances.

breaks: Array giving the thresholds applied on distances in order to derive the lags.

codir: Vector defining the direction for experimental structural tool.

size: Dimension of the variogram arrays

sw: Array containing the weights attached to each variogram lag

hh: Array containing the average distance attached to each variogram lag

gg: Array containing the average value of the structural tool attached to each variogram lag

12.2.2 Accessors

These are the different read/write accessors of some variables in the class:

vario$calcul: Type of calculation of the experimental structural tool.

vario$by.sample: Way the variogram calculation is performed.

vario$ndim: Space dimension.

vario$nvar: Number of variables.

vario$ndir: Number of directions for the calculation of experimental structural tools (not allowed for writing)

vario$opt.code: Option concerning the use of the code during the calculation of the experimental structural tool.

vario$tolcode: Tolerance for the use of the code during the calculations.

vario$means: Array of means of the variables.

vario$vars: Array of variances of the variables.

vario$vardirs: Array of directional variogram calculations (not defined for writing).

These are the different read/write accessors of some variables in the *Vardir* class:

vardir$npas: Number of lags.

vardir$npatot: Number of calculated values.

vardir$pas: Lag value.

vardir$tol: Tolerance on distance.

vardir$flag.regular: Flag specifying if regular distances are considered or if lags are calculated using thresholds defined using the *breaks* parameter.

vardir$codir: Direction definition vector

vardir$size: Dimension of the variogram arrays

vardir$sw: Array of weights (not defined for writing)

vardir$hh: Array of distances (not defined for writing)

vardir$gg: Array of variogram values (not defined for writing)

These are the different read/write accessors of some arrays in the class:

vario[i,j,k,l]: The experimental structural tool in the direction i, for the pair of variables j and k and for the lag l. The output is a list composed of three element: sw (weight), hh (distance) and gg (structural tool).

vario[i,j,k]: The experimental structural tool in the direction i, for the pair of variables j and k, for all the lags. The output is a structure of class *vardir*.

vario[i]: The experimental structural tool in the direction i, for all pairs of variables and for lags. The output is a structure of class *vardir*.

These are the different read/write accessors of some arrays in the class *vardir*:

vardir[i,j,k]: The experimental structural tool for the pair of variables i and j and for the lag k. The output is a list composed of three element: sw (weight), hh (distance) and gg (structural tool).

vardir[i,j]: The experimental structural tool for the pair of variables i and j, for all the lags. The output is a structure of class *vardir*.

12.2.3 Generic functions

vario.plot: Represent the contents of a experimental structural tool graphically. This function corresponds to the generic command *plot*.

vario.print: Print the contents of an object belonging to the *Vario* class. This function corresponds to the generic command *print*.

vario.read: Create a new Vario object by reading the contents of an ASCII file according to a format specific to the RGeostats package. This function corresponds to the generic command *ascii.read*.

vario.write: Write the contents of a Vario object into an ASCII file according to a format specific to the RGeostats package. This function corresponds to the generic command *ascii.write*.

12.2.4 Utilities

These utilities are specific to the Vario class. They will be merely described in this manual. The interested user will use the on-line help for more information. Note that the specific functions used to calculate the variograms in different conditions will not be described in this paragraph:

vario.window: Return information on the graphic window containing the graphic representation of an experimental variogram. This information essentially concerns the extension of the window and its possible dilation.

vario.model.check: Check if an object of the class Model and an object of the class Vario are compatible. This test relies on the space dimension and, optionally, on the number of variables.

12.2.5 ASCII format

Format of the ASCII file for an object of the *vario* class:

- Dimension of the space
- Number of variables
- Number of directions where the experimental variogram is calculated
- Option for the code selection:

  0: no selection performed on the code

  1: samples are compared only if the distance between their codes is smaller (or equal) than the tolerance for code selection

  2: samples are compared only if their codes are different

- Tolerance for code selection (only used if the option for the code selection is set to 1)
- Scaling factor (only used for the transitive covariogram)

For each direction, the following parameters:

- Flag for regular lags
- Number of lags
- Lag value
- Tolerance on the angles
- Direction coefficients: vector whose dimension is equal to the space dimension
- Direction increments defined as increments on the grid: vector whose dimension is equal to the space dimension

If the experimental variograms, the following information is dumped out for each lag:

- the weight attached to the lag (usually the number of pairs)
- the average distance
- the variogram value

12.3 Model

The Model class contains the information of the structural model composed of one or several basic structures. Note that the drift information (which is usually attached to the Model definition) is not stored within an object of the class Model.

The model is the necessary ingredient for geostatistical procedures such as kriging or simulations, as it describes the structural characteristics for any distance and any direction in the space.

12.3.1 Field

An object of the class *Model* contains the following fields:

ndim: Space dimension

nvar: Number of variables

basics: An array of objects belonging to the class *melem* used to describe the basic structures and described below.

An object of the class *Melem* contains the following fields:

vartype: Type of basic structure. In RGeostats, their is a large number of basic structures available: however, not that all basic structures are not available in any case: in particular, some of them are limited to certain range of space dimension; others are not available depending on the degree of intrinsicity of the model. The following list shows the available basic structures together with their corresponding names:

Nugget Effect: nugget effect component

Exponential: exponential structure

Spherical: Spherical structure

Cubic: Cubic structure

Gaussian: Gaussian structure

Cardinal Sine: Cardinal Sine structure

J Bessel: Structure corresponding to the J Bessel function

K Bessel: Structure corresponding to the K Bessel function

Gamma: Gamma structure

Cauchy: Cauchy structure

Stable: Stable structure

Linear: Linear structure (used in intrinsic case or in cases with drift)

Power: Power structure

Order-1 GC: Generalized Covariance of order 1 (only used in the intrinsic case and in cases with drift)

Order-3 GC: Generalized Covariance of order 3 (only used for cases with linear or higher degree drifts)

Order-5 GC: Generalized Covariance of order 5 (only used for cases with quadratic or higher degree drifts)

Exp2dfact: Factorized structure with an Exponential in 2-D and an exponential along the third direction

Expfact: Factorized exponential structure

Reg1D: variogram with hole effect obtained from the residuals of a random function with a linear variogram to which a moving average is subtracted

sill: Array of sills for the current basic structure. This is a square matrix, with a dimension equal to the number of variables), which must be definite positive. In the monovariate case, this sill is a single value for each basic structure. The term *sill* is generic as it contains any multiplicative coefficient, regardless of the fact that the covariance actually presents a genuine sill: in particular, this parameter contributes to the slope value in the case of linear variogram.

range: Range of the structure. The term *range* is generic as it corresponds to a scaling factor for the distances, regardless of the fact that the covariance actually presents a genuine range: in particular, this parameter contributes to the slope value in the case of linear variogram. Note that the range actually corresponds to the practical range (rather than the theoretical one).

param: Third parameter necessary for some special basic structures (such as power model for example)

flag.aniso: Flag which tells if some anisotropy must be considered or not. This flag, when FALSE, shortens the calculations and gains computing time.

aniso.rotmat: Array which gives the rotation matrix used in an anisotropic case (flag.aniso = TRUE). Its dimension is equal to the space dimension.

aniso.coeffs: Array which gives the anisotropy coefficients that divides the (isotropic) *range* in order to obtain the anisotropic range along each direction (after rotation).

For more information on the models, please refer to the Model Appendix.

12.3.2 Accessors

These are the different read/write accessors of some variables in the *model* class:

model$ndim: Space dimension.

model$nvar: Number of variables.

model$ncova: Number of basic structures (not available for writing).

model$basics: Array of objects of class *Melem* containing the characteristics of a basic structure.

These are the different read/write accessors of some variables in the *melem* class:

melem$vartype: Type of the basic structure.

melem$sill: Array of sills (or slopes).

melem$range: Range or scale factor.

melem$param: Third parameter.

melem$flag.aniso: Flag telling if the anisotropy is required.

melem$aniso.rotmat: Anisotropy rotation matrix

melem$aniso.coeffs: Array of anisotropy coefficients.

This is the read/write accessor of an array in the class:

model[i]: The object of the *melem* class containing the basic structure *i*.

12.3.3 Generic functions

model.plot: Represent the contents of a model graphically. This function corresponds to the generic command *plot*.

model.print: Print the contents of an object belonging to the *Model* class. This function corresponds to the generic command *print*.

model.read: Create a new object belonging to the *Model* class by reading the contents of an ASCII file according to a format specific to the RGeostats package. This function corresponds to the generic command *ascii.read*.

model.write: Write the contents of an object of the *Model* class into an ASCII file according to a format specific to the RGeostats package. This function corresponds to the generic command *ascii.write*.

12.3.4 Utilities

These utilities are specific to the *Model* class. They will be merely described in this manual. The interested user will use the on-line help for more information. Note that the different specific functions used to fit the model will not be described in this paragraph:

model.eval: Evaluate the Model for a given set of variables, a given direction and a given set of distances.

model.extend: Convert a monovariate model into a multivariate one.

model.grid: Create a new grid with a variable containing the value of the model evaluated at each grid node.

model.input: Define the model interactively.

model.pgs: Given the model of the underlying Random Gaussian variable and an experimental variogram of indicators, evaluate the theoretical variogram according to the PGS method.

model.sample: Sample a model along the lags of an experimental variogram specified in an object of the *Vario* class.

model.window: Return information on the graphic window containing the graphic representation of a model. This information essentially concerns the extension of the window and its possible dilation.

12.3.5 ASCII format

Format of the ASCII file for an object of the *model* class:

- Dimension of the space
- Number of variables
- Dimension of the field (used to build the stationary equivalent covariance of generalized covariance)
- Radius of the convolution ball (only used for gradient calculation)
- Number of basic structures (covariance, variogram or generalized covariance)
- Number of basic drift functions

For each basic structure:

- the type (see the appendix for the list of basic structures)

- the range. In the anisotropic case, this value will serve as the reference *isotropic range*.
- the third parameter (it must be provided even if not necessary for the given structure)

For each basic drift function:

- the type (see the appendix for the list of the drift functions)

Per basic structure:

- the (square) matrix of sills which should be definite positive: its dimension is equal to the number of variables

12.4 Neigh

The Neigh class contains the information of the neighborhood which defines the selection of the active samples used in the estimation or the simulation of a target site.

This neighborhood belongs to one of the following three characteristics:

- Unique neighborhood where all samples are systematically selected for each target site
- Moving Neighborhood where only a selection of active samples is chosen which *moves* with the location of the target site
- other neighborhoods which are defined specifically for some applications, such as :
  - *Image* Neighborhood used when processing information defined on a Regular Grid
  - *Be*nch Neighborhood where all the samples belonging to a horizontal slice (called a bench) are systematically selected for all the target sites which belong to this slice.

12.4.1 Fields

An object of the class *Neigh* contains the following fields:

ndim: Space dimension

type: Type of neighborhood according to the following list:

0: Unique Neighborhood

1: Bench Neighborhood

2: Moving Neighborhood

The subsequent parameters are valid only in the case of *Moving* Neighborhood.

flag.sector: Specify if the Moving Neighborhood must be performed selecting samples according to the sector to which they belong

flag.aniso: Specify if the Moving Neighborhood must be considered as anisotropic. The distance from each sample to the target site must be converted into an isotropic equivalent distance before the selection algorithm is applied.

flag.rotation: Specify if the Anisotropy of the Moving Neighborhood must be rotated.

nmini: Minimum number of samples in the Neighborhood. If this number is not reached, the target site is not processed.

nmaxi: Maximum number of samples in the Neighborhood. This value serves for dimensioning the arrays.

nsect: Number of angular sectors (only if *flag.sector* is set to TRUE).

nsmax: Maximum number of samples selected per angular sector (only if *flag.sector* is set to TRUE).

dmax: Maximum isotropic distance

coeffs: Array of anisotropic coefficients (only used if *flag.aniso* is set to TRUE)

rotmat: Rotation anisotropy matrix (only used if *flag.aniso* is set to TRUE)

The subsequent parameter is valid only in the case of *Bench* Neighborhood:

width: Width of the vertical bench

The subsequent parameters are valid only in the case of *Image* Neighborhood:

radius: Radius of the image neighborhood (defined in terms of grid nodes)

skip: Skipping ratio for the selection of the Image Neighborhood. For example, in an Image Neighborhood in 2-D with a radius of 5, the number of grid nodes contained in a Neighborhood is equal to 121. For a skipping ratio of 1, all 121 samples are used; for a skipping ratio of 2, only 61 samples are selected.

12.4.2 Accessors

These are the different read/write accessors of some variables in the *neigh* class:

neigh$ndim: Space dimension

neigh$type: Type of the Neighborhood:

- **0:** Unique Neighborhood
- **1:** Bench Neighborhood
- **2:** Moving Neighborhood

neigh$flag.sector: Tells if the Moving Neighborhood search uses sectors

neigh$flag.aniso: Tells if the Moving Neighborhood uses anisotropic distance

neigh$flag.rotation: Tells if the anisotropy of the Moving Neighborhood is defined in a rotated system

neigh$nmini: Minimum number of samples in the Moving Neighborhood

neigh$nmaxi: Maximum number of samples in the Moving Neighborhood

neigh$nsect: Number of angular sectors

neigh$nsmax: Maximum number of selected samples per angular sector

neigh$width: Width of the bench for Bench Neighborhood

neigh$dmax: Maximum isotropic distance

neigh$coeffs: Array of anisotropic coefficients

neigh$rotmat: Rotation matrix for the anisotropy

neigh$radius: Radius for the Image Neighborhood

neigh$skip: Skipping ratio for the Image Neighborhood

12.4.3 Generic functions

neigh.print: Print the contents of an object belonging to the *Neigh* class. This function corresponds to the generic command *print*.

neigh.plot: Represent the contents of an object of the *Neigh* class graphically. This function corresponds to the generic command *plot*.

neigh.read: Create a new object of the *Neigh* class by reading the contents of an ASCII file according to a format specific to the RGeostats package. This function corresponds to the generic command *ascii.read*.

neigh.write: Write the contents of an object of the *Neigh* class into an ASCII file according to a format specific to the RGeostats package. This function corresponds to the generic command *ascii.write*.

12.4.4 Utilities

These utilities are specific to the *Neigh* class. They will be merely described in this manual. The interested user will use the on-line help for more information.

neigh.init: Create an object of the class *Neigh* according to one of the possible types.

neigh.input: Create an object of the *Neigh* class interactively

12.4.5 ASCII format

Format of the ASCII file for an object of the *neigh* class:

- Type of Neighborhood:

  0: Unique neighborhood
  1: Bench neighborhood
  2: Moving neighborhood
  3: Image neighborhood

For the Unique Neighborhood, no other parameter is required.

For the Bench Neighborhood:

- Cross-validation flag: 1 to switch this option ON; 0 otherwise
- Width of the bench along the last space dimension (for example: along the third coordinate in the 3-D case)

For the Moving Neighborhood:

- Cross-validation flag: 1 to switch this option ON; 0 otherwise
- Flag for a search using angular sectors
- Minimum number of samples below which the treatment is not performed
- Maximum number of samples in the neighborhood
- Number of angular sectors
- Maximum number of samples per angular sector
- Maximum isotropic radius of the neighborhood

For the Image Neighborhood:

- Cross-validation flag: 1 to switch this option ON; 0 otherwise
- Skipping ratio
- Radius of the Image Neighborhood (0: central node)

12.5 Anam

The *Anam* class describes the Gaussian Anamorphosis Transform which enables the transformation of a variable from a raw (variable denoted Z) to a)normalized) gaussian scale (variable denoted Y), and vice-versa. This transform is captured as a polynomial expansion using (a limited set of) Hermite polynomials. For the back transform (from gaussian to raw scale), the raw variable is assumed to be bounded.

12.5.1 Fields

An object of the class *Anam* contains the following fields:

nh: Number of Hermite polynomials in the expansion

pymin, pymax: Practical bounds in the Gaussian scale

pzmin, pzmax: Practical bounds in the Raw scale

aymin, aymax: Absolute bounds in the Gaussian scale

azmin, azmax: Absolute bounds in the Raw scale

variance: Variance of the data (used to scale the Gaussian Anamorphosis Transform)

psi: Array of coefficients of the Hermite polynomials

12.5.2 Accessors

These are the different read/write accessors of some variables in the *anam* class:

anam$nh: Number of Hermite polynomials

anam$pymin, anam$pymax: Practical bounds in the Gaussian scale

anam$pzmin, anam$pzmax: Practical bounds in the Raw scale

anam$aymin, anam$aymax: Absolute bounds in the Gaussian scale

anam$azmin, anam$azmax: Absolute bounds in the Raw scale

anam$variance: Variance of the data

anam$psi: Array of coefficients of the Hermite polynomials

This is the read/write accessor of an array in the *Anam* class:

anam[i]: The coefficient of the i^{th} Hermite polynomial.

12.5.3 Generic functions

anam.print: Print the contents of an object belonging to the *Anam* class. This function corresponds to the generic command *print*.

anam.read: Create a new object of the *Anam* class by reading the contents of an ASCII file according to a format specific to the RGeostats package. This function corresponds to the generic command *ascii.read*.

anam.write: Write the contents of an object of the *Anam* class into an ASCII file according to a format specific to the RGeostats package. This function corresponds to the generic command *ascii.write*.

12.5.4 Utilities

These utilities are specific to the *Anam* class. They will be merely described in this manual. The interested user will use the on-line help for more information:

anam.fit: Fit the Gaussian Anamorphosis starting from a Raw variable.

anam.y2z: Transform a Gaussian variable into a Raw variable, using the Anamorphosis Transform function stored in an object of the *Anam* class.

anam.z2y: Transform a Raw variable into a Gaussian variable, using the Anamorphosis Transform function stored in an object of the *Anam* class.

12.5.5 ASCII format

Format of the ASCII file for an object of the *anam* class:

- Number of Hermite polynomials
- Minimum absolute value for Z
- Maximum absolute value for Z
- Minimum absolute value for Y
- Maximum absolute value for Y
- Minimum practical value for Z
- Maximum practical value for Z
- Minimum practical value for Y
- Maximum practical value for Y

- Flag for Storage of Calculation Results: if 1, the coefficients of the Hermite polynomials are printed next

If calculations are dumped out:

- Calculated variance
- Coefficients of the Hermite polynomials : vector of dimension equal to the number of Hermite polynomials)

12.6 Polygon

An object of the *Polygon* class contains one or several *Polysets*. A *Polyset* if a 2-D closed polyline which is used to:

- select samples lying within the Polyset
- delineate a domain where the average of the variable must be estimated

12.6.1 Fields

An object of the *Polygon* class presents the following fields:

sets: An array of objects belonging to the class *Polyset* used to describe the basic closed polyline.

An object of the class *Polyset* contains the following fields:

x: Array of coordinates along X of the polyline vertices

y: Array of coordinates along Y of the polyline vertices

12.6.2 Accessors

These are the different read/write accessors of some variables in the *Polygon* class:

polygon$npol: Number Polysets contained in the Polygon

polygon$surface: Surface of the Polygon

polygon$xlim: Minimum and maximum coordinates of the polygon along X

polygon$ylim: Minimum and maximum coordinates of the polygon along Y

This is the read/write accessor of an array in the *Polygon* class:

polygon[i]: The i^{th} Polyset

polygon[i,j]: the 2-D coordinates of the j^{th} vertex of the i^{th} Polyset.

12.6.3 Generic functions

polygon.plot: Represent the contents of an object of the *Polygon* class graphically. This function corresponds to the generic command *plot*.

polygon.print: Print the contents of an object belonging to the *Polygon* class. This function corresponds to the generic command *print*.

polygon.read: Create a new object of the *Polygon* class by reading the contents of an ASCII file according to a format specific to the RGeostats package. This function corresponds to the generic command *ascii.read*.

polygon.write: Write the contents of an object of the *Polygon* class into an ASCII file according to a format specific to the RGeostats package. This function corresponds to the generic command *ascii.write*.

12.6.4 Utilities

These utilities are specific to the *Polygon* class. They will be merely described in this manual. The interested user will use the on-line help for more information:

polygon.create: Create a new Polygon or add a new Polyset to an already existing Polygon.

polygon.digit: Digitize one or several new Polysets from a plot, in order to produce a new Polygon.

polygon.inside: Checks if a set of points (characterized by the vectors of first two coordinates) belong to a Polygon or not.

polygon.projec: Apply the current projection to the vertices of a Polygon.

polygon.start: Attach a Polygon to be used by a C code. This function should only be used prior to a call to C code which expects a Polygon: the Polygon will be connected using the slot number returned by the function *polygon.start*.

12.6.5 ASCII format

Format of the ASCII file for an object of the *polygon* class:

- Number of Polysets in the Polygon

For each Polyset:

- Number of vertices
- 2-D Coordinates of the polyset vertices

12.7 Thresh

An object of the *Thresh* class contains a set of threshold intervals. Each interval is constituted by its two bounds, defined or not.

12.7.1 Fields

An object of the *Thresh* class presents the following fields:

nclass: Number of intervals

bounds: Matrix defining the lower and upper bounds of the intervals

12.7.2 Accessors

These are the different read/write accessors of some variables in the *Thresh* class:

thresh$nclass: Number of intervals

thresh$bounds: Matrix of bounds

This is the read/write accessor of an array in the *Thresh* class:

thresh[i,j]: The j^{th} bound of the i^{th} interval. The argument i is limited to the number of intervals ($nclass$) and the argument j is limited to 2

thresh[,j]: The vector of the j^{th} bound for all intervals.

thresh[i,]: The bounds of the i^{th} interval.

thresh[]: The matrix of bounds.

12.7.3 Generic functions

thresh.print: Print the contents of an object belonging to the *Thresh* class. This function corresponds to the generic command *print*.

12.7.4 Utilities

These utilities are specific to the *Thresh* class. They will be merely described in this manual. The interested user will use the on-line help for more information:

thresh.input: Define a new object of the *Thresh* class interactively.

12.8 Rule

An object of the *Rule* class contains the lithotype rule which enables the translation of a set of underlying random gaussian variables (GRF) into a categorical variable presenting several facies.

The lithotype rule is represented for two GRFs: this 2-D space is subdivided into as many rectangular areas as facies, which constitute a partition of 2-D. Note that, to define the areas for n facies, it suffices to define *n-1* edges (or nodes).

12.8.1 Fields

An object of the *Rule* class presents the following fields:

nbnode: Number of nodes (equal to the number of facies - 1)

mode.rule: Type of Lithotype Rule definition:

- **0:** Standard Lithotype Rule
- **1:** Shift: the second GRF is a shifted version of the first GRF
- **2:** Shadow: a shadow is applied to the first GRF

rho: Correlation between the two underlying GRFs (only used for standard Lithotype Rule)

slope: Slope for the shadow calculation (only used for the shadow option)

dinf: Elevation of the plane where the shadow is calculated (only used for the shadow option)

dsup: Maximum elevation above which the relief is truncated before calculating its shadow (only used for the shadow option)

shift: Value for the shift (only used for the shift option)

nodes: Array for the characterization of the nodes

The array *nodes* provides the definition of the *nbnode* nodes. They are defined as a nested list. Each node corresponds to a vector of the following six values:

0: Type of the parent node

1: Rank of the parent node

2: Orientation with respect to the parent node

3: Type of the current node

4: Rank of the current node

5: Facies value

A node type may be one of the following values:

0: for a node defining a facies

1: for a node defining a threshold along the first GRF

2: for a node defining a threshold along the second GRF

The orientation (with respect to the parent node) is:

1: if the current node concerns values of the GRF smaller than the threshold corresponding to the parent node

2: if the current node concerns values of the GRF larger than the threshold corresponding to the parent node

12.8.2 Accessors

These are the different read/write accessors of some variables in the *Rule* class:

rule$nbnode: Number of nodes

rule$mode.rule: Type of Lithotype Rule

rule$rho: Correlation coefficient between the two underlying GRFs

rule$slope: Slope (for the Shadow Lithotype Rule)

rule$dinf: Plane elevation (for the Shadow Lithotype Rule)

rule$dsup: Relief truncation (for the Shadow Lithotype Rule)

rule$shift: Shift (for the Shifted Lithotype Rule)

rule$nodes: Array defining the node characteristics

12.8.3 Generic functions

rule.print: Print the contents of an object belonging to the *Rule* class. This function corresponds to the generic command *print*.

rule.plot: Represent the contents of an object of the *Rule* class graphically. This function corresponds to the generic command *plot*.

rule.read: Create a new object of the *Rule* class by reading the contents of an ASCII file according to a format specific to the RGeostats package. This function corresponds to the generic command *ascii.read*.

rule.write: Write the contents of an object of the *Rule* class into an ASCII file according to a format specific to the RGeostats package. This function corresponds to the generic command *ascii.write*.

12.8.4 Utilities

These utilities are specific to the *Rule* class. They will be merely described in this manual. The interested user will use the on-line help for more information:

rule.input: Define a new object of the *Rule* class interactively.

12.8.5 ASCII format

Format of the ASCII file for an object of the *rule* class:

- Mode used for the definition of the Lithotype Rule:

  0: Standard option

  1: Shift Rule definition

  2: Shadow Rule definition

- The correlation coefficient between the two underlying gaussian random functions (only used in the standard case)

The next parameters are only used for the "Shadow" option (although they must be defined anyway):

- The slope of the shadow
- The lower truncation value
- The upper truncation value

The next parameters are only used for the "Shift" and the "Shadow" options (although they must be defined anyway):

- The shift value along X
- The shift value along Y
- The shift value along Z

The rest of this file contains the definition of the different nodes used to define the Lithotype Rule. We must first define:

- The number of subsequent nodes

Per node, we must define:

- The type of the parent node
- The rank of the parent node
- The orientation of the parent node
- The type of the node:

  0: Facies)

  1: Threshold along Y1

  2: Threshold along Y2

- The rank of the new node (starting from 1)
- The rank of the facies

12.9 Tokens

An object of the *Tokens* class contains the characteristics of the families of tokens used when performing an Object Based Simulation. Each family of tokens contain a variable number of parameters depending on its type, as well as its proportion.

12.9.1 Fields

An object of the *Tokens* class presents the following fields:

nbtokens: Number of token families

nbparams: Total number of parameters describing the geometry of the tokens

types: List of the token types

props: Array giving the proportion for each token family. These proportions should add up to 1.

mean: Array of centers for the parameters defining the geometry of the tokens

stdev: Array of radii for the parameters defining the geometry of the tokens.

12.9.2 Generic functions

token.print: Print the contents of an object belonging to the *Token* class. This function corresponds to the generic command *print*.

12.9.3 Utilities

These utilities are specific to the *Tokens* class. They will be merely described in this manual. The interested user will use the on-line help for more information:

token.input: Define a new object of the *Token* class interactively.

13 The graphics

This paragraph gives the general information on the graphics used in RGeostats.

A first remarks is that each graphic page contains several scenes by default: some applications may only use one figure (plot of a db) but other applications benefit from this multiple scenes behavior (plot a multivariate variogram or model). The technique used for this multiple scenes (i.e. *split.screen*) is incompatible with the other multi-screen procedure such as *mfrow*.

The multiple scene organization is compatible with another split of the screen already defined by the user. For this sake, all the graphic procedures of RGeostats provide the *reset* parameter:

- if *reset=TRUE* (default value), any graphic will first erase any already existing page subdivision
- if *reset=FALSE*, the current page subdivision is kept and the current graphic (which may itself be subdivided) is produced in the current scene.

Appendices

14 Model definition

The package RGeostats offers a list of basic structures that can be used in order to construct a Model. Each basic structure is now described with its exact formula.

We recall that:

- the basic structure includes covariances, variograms or generalized covariances
- a covariance is a particular variogram (bounded), a variogram (and therefore a covariance0 is a particular generalized covariance
- a basic structure is valid for certain space dimensions
- in all subsequent formulae, the value h defines the modulus of the (isotropic) distance: therefore, this distance is always positive
- some covariance use a practical range which corresponds to the distance beyond which the covariance reaches 95% of the sill value

14.1 List of the Basic structures

14.1.1 Nugget Effect

This is a covariance, defined for any space dimension:

$$C(h) = C_0 \delta(h)$$

where:

- C_0 is the sill
- $\delta(h)$ is a function which returns 1 if $h = 0$, and 0 for strictly positive distance

14.1.2 Exponential

This is a covariance, defined for any space dimension:

$$C(h) = C \times \exp\left(-\frac{h}{a/s}\right)$$

where:

- C is the sill
- a is the (practical) range
- $s = 2.995732$

14.1.3 Spherical

This is a covariance, defined up to the third space dimension:

$$C(h) = C \times \left[1 - \frac{3}{2}\frac{h}{a} + \frac{1}{2}\left(\frac{h}{a}\right)^3\right]$$

where:

- C is the sill
- a is the range

14.1.4 Gaussian

This is a covariance, defined for any space dimension:

$$C(h) = C \times \exp\left(-\left(\frac{h}{a/s}\right)^2\right)$$

where:

- C is the sill
- a is the (practical) range
- $s = 1.730818$

14.1.5 Cubic

This is a covariance, defined up to the third space dimension:

$$C(h) = C \times \left[1 - 7h^2 + \frac{35h^3}{4} - \frac{7h^5}{2} + \frac{3h^7}{4}\right]$$

where:

- C is the sill
- a is the range

14.1.6 Cardinal Sine

This is a covariance, defined for any space dimension:

$$C(h) = C \times \left[\frac{\sin\left(\frac{h}{a/s}\right)}{\frac{h}{a/s}}\right]$$

where:

- C is the sill
- a is the (practical) range
- $s = 20.371$

14.1.7 J-Bessel

14.1.8 K-Bessel

14.1.9 Gamma

This is a covariance, defined for any space dimension:

$$C(h) = C \times \left[\frac{1}{\left(1 + \frac{h}{a}\right)^{\alpha}}\right]$$

where:

- C is the sill
- a is the range
- α is the (positive) exponent defined as the third parameter

14.1.10 Cauchy

This is a covariance, defined for any space dimension:

$$C(h) = C \times \left[\frac{1}{\left(1 + \left(\frac{h}{a}\right)^2\right)^{\alpha}} \right]$$

where:

- C is the sill
- a is the range
- α is the (positive) exponent defined as the third parameter

14.1.11 Stable

This is a covariance, defined for any space dimension:

$$C(h) = C \times \exp\left(-\left(\frac{h}{a}\right)^{\alpha}\right)$$

where:

- C is the sill
- a is the range
- α is the exponent defined as the third parameter which lies within $[0; 2]$

14.1.12 Linear

This is a variogram, defined for any space dimension:

$$\gamma(h) = C \times \frac{h}{a}$$

where:

- C is the multiplicative coefficient (also called *sill* in the interface)
- a is the scale factor (also called *range* in the interface)

14.1.13 Power

This is a variogram, defined for any space dimension:

$$\gamma(h) = C \times \left(\frac{h}{a}\right)^{\alpha}$$

where:

- C is the multiplicative coefficient (also called *sill* in the interface)
- a is the scale factor (also called *range* in the interface)
- α is the exponent defined as the third argument which must lie within $[0; 2]$

14.1.14 Order-1 GC

This is a generalized covariance defined for an intrinsic random function, defined for any space dimension:

$$\gamma(h) = C \times \left(\frac{h}{a}\right)$$

where:

- C is the multiplicative coefficient (also called *sill* in the interface)
- a is the scale factor (also called *range* in the interface)

14.1.15 Order-3 GC

This is a generalized covariance defined for a first order random function (it needs a first order polynomial drift), defined for any space dimension:

$$K(h) = C \times \left(\frac{h}{a}\right)^{3}$$

where:

- C is the multiplicative coefficient (also called *sill* in the interface)
- a is the scale factor (also called *range* in the interface)

14.1.16 Spline GC

This is a generalized covariance defined for a first order random function (it needs a first order polynomial drift), defined for any space dimension:

$$K(h) = C \times \left(\frac{h}{a}\right)^2 \ln\left(\frac{h}{a}\right)$$

where:

- C is the multiplicative coefficient (also called *sill* in the interface)
- a is the scale factor (also called *range* in the interface)

14.1.17 Order-5 GC

This is a generalized covariance defined for a second order random function (it needs a second order polynomial drift), defined for any space dimension:

$$K(h) = C \times \left(\frac{h}{a}\right)^5$$

where:

- C is the multiplicative coefficient (also called *sill* in the interface)
- a is the scale factor (also called *range* in the interface)

14.1.18 Cosinus

This is a covariance, defined for a one dimension space:

$$C(h) = C \times \cos\left(\frac{2\pi h}{a}\right)$$

where:

- C is the sill
- a is the period

14.1.19 Triangle

This is a covariance, defined for a one dimension space:

$$C(h) = C \times \left(1 - \frac{h}{a}\right) \times \delta\left(h < a\right)$$

where:

- C is the sill
- a is the range
- $\delta(f)$ is a function which returns 1 if f is true and 0 otherwise

14.1.20 Cosexp

This is a covariance, defined for a one dimension space:

$$C(h) = C \times \cos\left(\frac{2\pi h}{\alpha}\right) \times \exp\left(-\frac{h}{a/s}\right)$$

where:

- C is the sill
- a is the (pratical) range
- $s = 2.995732$
- α is the period

14.1.21 Exp2dfact

This is a covariance, defined for any space dimension:

$$C(h) = C \times \exp\left(-\frac{h_{2D}}{a_{2D}/s}\right) \times \prod \exp\left(\frac{h_i}{a_i/s}\right)$$

where:

- C is the sill
- a is the (practical) range
- $s = 2.995732$
- h_{2D} refers to the distance in the 2-D plane
- h_i refers to the distance in any subsequent space dimension

14.1.22 Expfact

This is a covariance, defined for any space dimension:

$$C(h) = C \times \prod \exp\left(-\frac{h_i}{a_i/s}\right)$$

where:

- C is the sill
- a is the (practical) range
- $s = 2.995732$
- h_i refers to the distance in any space dimension

14.1.23 Reg1D

This is a covariance, defined for 1 dimension only:

$$\begin{array}{cc} C(h) = C \times \left[1 - 3\frac{h}{a}\left(1 - \frac{h}{2a}\left(1 + \frac{h}{6a}\right)\right)\right] & h < 0.5 \\ C(h) = C \times \left[-2 + 3\frac{h}{a}\left(1 - \frac{h}{2a}\left(1 - \frac{h}{6a}\right)\right)\right] & 0.5 < h < 1 \\ C(h) = 0 & h > 1 \end{array}$$

where:

- C is the sill
- a is the range
- h refers to the distance in 1 dimension

14.1.24 Pentamodel

This is a covariance, which corresponds to the spherical model calculated in R^7, after fourth order "montée" (upscaling)

$$\begin{array}{cc} C(h) = C \times \left[1 - \frac{22}{3}\left(\frac{h}{a}\right)^2 + 33\left(\frac{h}{a}\right)^4 - \frac{77}{2}\left(\frac{h}{a}\right)^5 + \frac{33}{2}\left(\frac{h}{a}\right)^7 - \frac{11}{2}\left(\frac{h}{a}\right)^9 + \frac{5}{6}\left(\frac{h}{a}\right)^{11}\right] & h < 1 \\ C(h) = 0 & 1 < h \end{array}$$

where:

- C is the sill
- a is the range
- h refers to the isotropic distance

14.1.25 Spline-2

This is a generalized covariance defined for a first order random function (it needs a second order polynomial drift), defined for any space dimension:

$$K(h) = C \times \left(1 - 2\left(\frac{h}{a}\right)^2 + \left(\frac{h}{a}\right)^3\right) \quad h < a$$

where:

- C is the multiplicative coefficient (also called *sill* in the interface)
- a is the scale factor (also called *range* in the interface)

14.2 Anisotropy

This paragraph describes the way RGeostats handles the anisotropy.

We first recall that a model is a combinaison of several basic structures. Each basic structure corresponds to one of the structures listed and illustrated in the previous paragraph. Each basic structure (at least the ones which require a range definition) can be anisotropic.

Traditionally, two sorts of anisotropies can be distinguished:

- the geometric anisotropy: the ranges are different in different directions, so that a simple stretch of the space (in the relevant direction) would bring the structure back to isotropy.
- the zonal anisotropy: in one given direction of the space, the variability (corresponding to the sill) is smaller than in any other direction of the space.

First, let us note that both anisotropies require the definition of the rotation system in which the anisotropy will be expressed: we will then speak of the main anisotropy (orthogonal) directions.

14.2.1 Geometric anisotropy

When the main anisotropy directions have been defined, the geometric anaisotropy consists in defining the ranges along these different directions.

For the covariance evaluation, the generic formula (defined in the previous paragraph) is used. For example, the isotropic spherical covariance is defined in the 2-D space as:

$$C(h) = C \times \left[1 - \frac{3}{2}\frac{h}{a} + \frac{1}{2}\left(\frac{h}{a}\right)^3\right]$$

where:

- C is the sill
- a is the (isotropic) range

We now consider the anisotropic spherical covariance considering that the main anisotropy directions coincide with the main axes of the system (otherwise, we must simply perform the rotation beforehand). The ranges are defined in the two main anisotropy directions (therefore along X and Y), denoted as a_x and a_y.

The value $\frac{h}{a}$ of the general formula must be replaced by the weighted distance:

$$\sqrt{\left(\frac{h_x}{a_x}\right)^2 + \left(\frac{h_y}{a_y}\right)^2}$$

14.3 Zonal anisotropy

When the main anisotropy directions have been defined, we must defined the direction in which the variable shows no variability: say the Y direction for example. Then it can almost be considered as a geometric anisotropy where the range along the Y direction is set to a large arbitrary distance.

However, it is not trivial to enter such as a large distance. For that reason, in the interactive way to input a model (function *model.input*), it suffices to set conventionally the anisotropy range, in the Y direction, to NA. The range printed as N/A in the anisotropy direction confirms that this direction corresponds to the zonal anisotropy main direction.

```
Count of Basic structures [1,NA] : 1
1 - Nugget Effect
2 - Exponential
3 - Spherical
4 - Gaussian
5 - Cubic
6 - Cardinal Sine
7 - J-Bessel
8 - K-Bessel
9 - Gamma
10 - Cauchy
11 - Stable
12 - Linear
13 - Power
14 - Order-1 GC
15 - Cosexp
16 - Exp2dfact
17 - Expfact
Rank of the basic structure [1,17] :  2
Sill [0.000000,NA] : 1
Anisotropy (Def=n) [y,n] :  y
Anisotropy rotation (Def=n) [y,n] :  n
Anisotropic Ranges(1) [0.000000,NA] : 1
Anisotropic Ranges(2) [0.000000,NA] : NA
...
Model characteristics
=====================
Space dimension = 2
Number of variable(s) = 1
Number of basic structure(s) = 1
Number of drift function(s) = 1
Number of drift equation(s) = 1
Covariance Part
---------------
- Exponential
Range = 1.000
Sill = 1.000
Anisotropy =
[,1] [,2]
[1,] 1.000 N/A
Total Sill = 1.000
Drift Part
----------
Universality Condition
```

When reading model from an ASCII file using the *model.read* procedure, the range in the Y direction should be set to NA again. But the NA string cannot be read: instead, it must be replaced by the conventional string "-999.0", as demonstrated in the following ASCII file (corresponding to the model entered interactively above);

```
Model
2 1 0.000000 0.000000 # General parameters
1 # Number of basic covariance terms
1 # Number of drift terms
2 1.000000 0.000000 # Covariance characteristics
1 # Anisotropy Flag
1.000000 -999.0 # Anisotropy Coefficients
0 # Anisotropy Rotation Flag
0 # Drift characteristics
1.000000 # Matrix of sills
```

14.4 Model demonstration

A special demonstration script is provided (in the standard *demo* command) which enables the user to visualize the aspect of the different basic structures. The procedure generates graphic windows where all the basic structures (which correspond to covariances or variograms) are displayed (4 by page).

It suffices to launch the demonstration script by typing:

```
demo(RGeostats.model)
```

The script generates the following pages if variogram models:

Figure 1: Model demonstration: Page #1

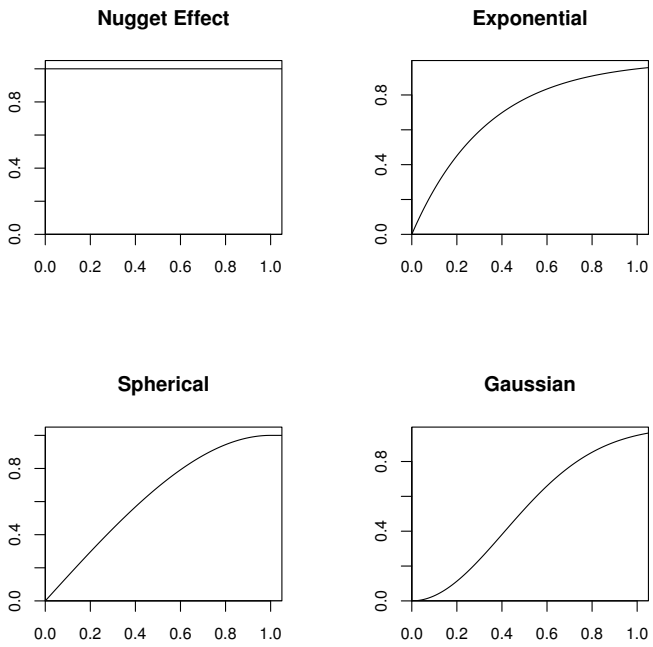


Figure 2: Model demonstration: Page #2

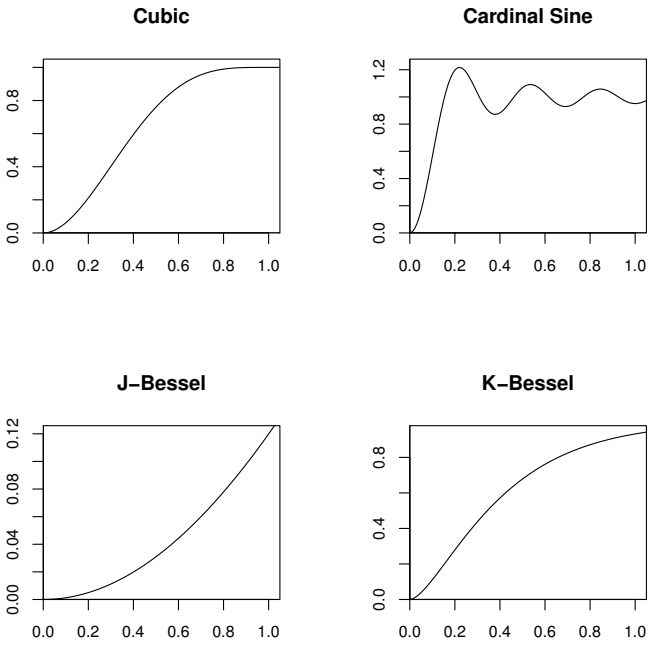


Figure 3: Model demonstration: Page #3

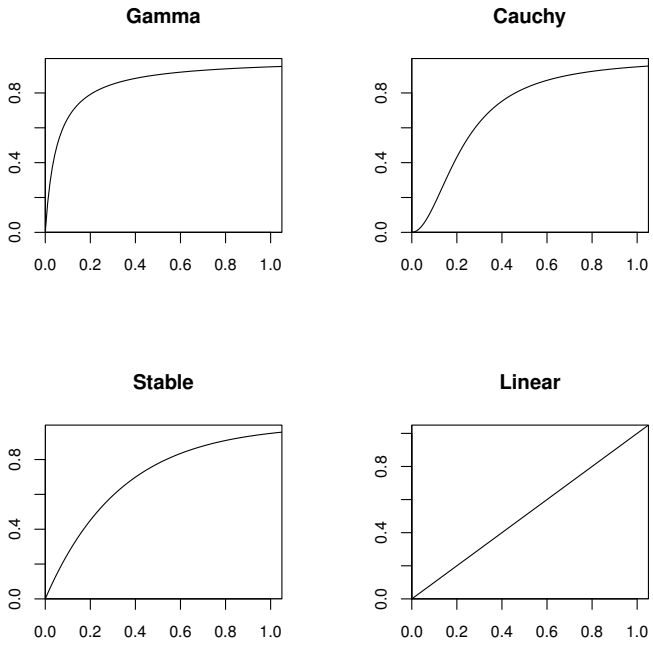


Figure 4: Model demonstration: Page #4

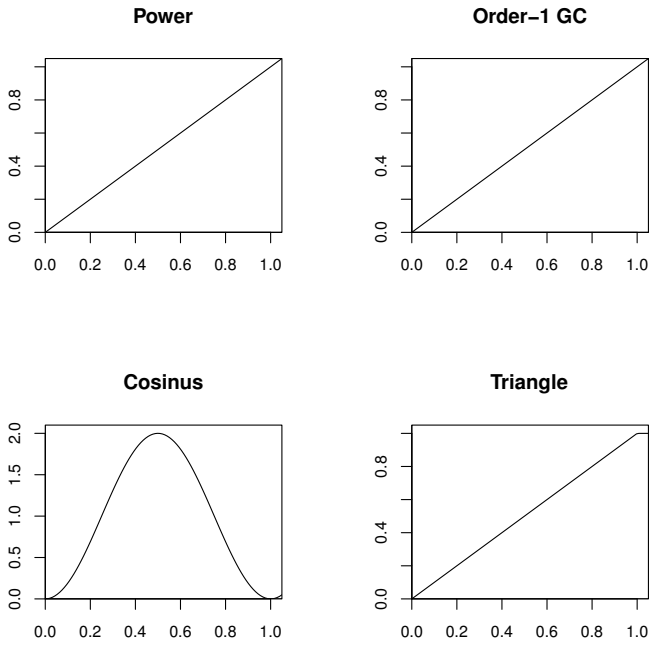


Figure 5: Model demonstration: Page #5

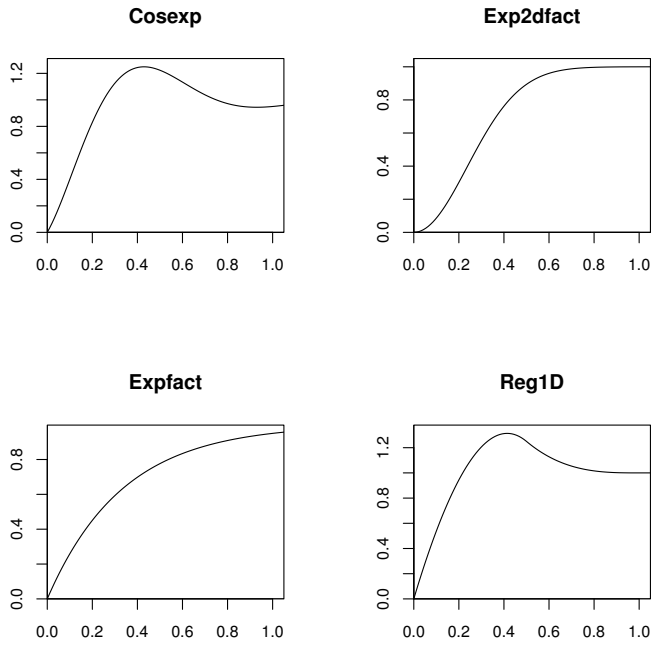


Figure 6: Model demonstration: Page #6

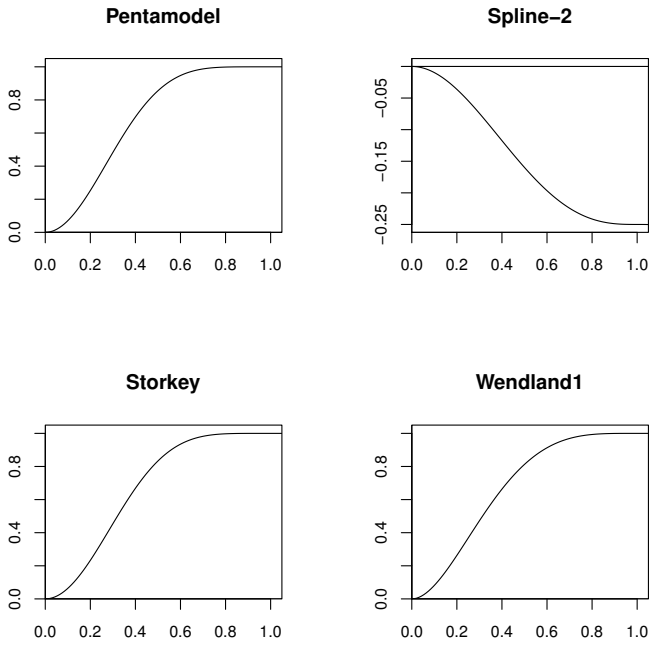
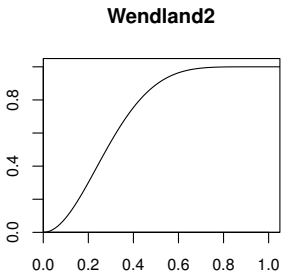


Figure 7: Model demonstration: Page #7



15 Projections

The package *RGeostats* handles the projections. In fact, the different classes and methods of this package are connected with a projection system, which itself is based on the *mapproj* and *mapproject* packages. The projection parameters are saved in the RGeostats Environment File, until they are either cancelled or modified.

Contents

List of Figures